Optimal Hiring of Cloud Servers

Andrew Stephen McGough[1] and Isi Mitrani[2]

[1] School of Engineering and Computing Sciences, Durham University, DH1 3LE, U.K.
stephen.mcgough@durham.ac.uk
[2] School of Computing Science, Newcastle University, NE1 7RU, U.K.
isi.mitrani@ncl.ac.uk

Abstract. A host uses servers hired from a Cloud in order to offer certain services to paying customers. It must decide dynamically when and how many servers to hire, and when to release them, so as to minimize both the job holding costs and the server costs. Under certain assumptions, the problem can be formulated in terms of a semi-Markov decision process and the optimal hiring policy can be computed. Two situations are considered: (a) jobs are submitted in random batches and servers can be hired for arbitrary periods of time; (b) jobs arrive singly and servers must be hired for fixed periods of time. In both cases, the optimal policies are compared with some simple and easily implementable heuristics.

1 Introduction

This paper focuses on certain special, and important, dynamic scheduling problems that arise in the market for computer services. It presents a general optimization methodology and applies it in situations where detailed exact and approximate solutions can be developed.

A host offers certain services which involve running user jobs. It does not own servers, but hires them on a temporary basis from a Cloud provider. The host must decide dynamically when, and how many, servers to hire. The objective is to manage optimally the long-term trade-offs between the operating costs (which depend on the number of servers hired), and the Quality-of-Service, or 'holding' costs (which are proportional to the number of jobs present).

Two distinct models are considered. In the first, jobs are submitted in batches of random size and at random intervals. Servers may be hired and released at arbitrary moments of time, hence the hiring decisions can be taken at the instants when new batches arrive. In the second model, servers must be hired for reasonably long fixed periods of time, e.g. by the hour. Hiring decisions are therefore assumed to take place at discrete moments in time, while jobs arrive and depart singly and in continuous time. The second model is perhaps closer to current practice, but the first one may come into its own since some Cloud providers are beginning to offer servers for very short-term hire, e.g. by the minute.

We show how, under certain assumptions, these dynamic optimization problems can be solved by formulating them in terms of semi-Markov decision processes and applying a policy improvement algorithm. The optimal hiring policy

A. Horváth and K. Wolter (Eds.): EPEW 2014, LNCS 8721, pp. 1–15, 2014.

can then be computed in a finite number of iterations. Although that computation is efficient, it may sometimes be too expensive to be carried out on-line. We therefore propose simple and easily implementable heuristic policies for both models. In numerical experiments, the performance of the heuristics is compared to that of the optimal policy.

An example of a company using Cloud servers is Cycle Computing[1], which acts as a broker offering virtual High-Throughput HTCondor [15] clusters in the Cloud. Different service facilities are also provided by interfaces such as e-Science Central [8], whereby access to Cloud computing resources is offered to users in a transparent manner. However, at present these systems do not make any attempt to optimize their operating policies.

The main distinguishing feature of the present study is that we carry out a rigorous dynamic optimization of the systems considered. That is, we consider operational decisions that depend not only on the system parameters, but also on the changing system state. Moreover, the optimization takes into account the transition probabilities between states, and hence covers a long-term system trajectory. This does not appear to have been done before.

Being able to determine the optimal operating policy is valuable, even when good heuristics exist. One may suspect that a simple heuristic policy will perform well, but the only way to quantify such a statement is to compute the optimal policy and carry out a proper comparison.

1.1 Related Work

The existing approaches to the server hiring problem are, on the whole, concerned with static policies. That is, the hiring decisions are based on knowing or estimating the characteristics of user demand. Those decisions change only when the demand parameters change. On the other hand, a dynamic policy reacts to random changes in the system state, even if the demand characteristics remain the same. In general, dynamic policies are more efficient than static ones, as we shall see when presenting our numerical results.

Mazzucco *et al.* [10] have used workload estimation in order to determine the optimal number of servers to hire. By assuming that impatient users will abandon job requests (common for HTTP) an Erlang-C problem is converted into an Erlang-A problem and a solution is obtained by a binary search algorithm. This work is extended in [12] to evaluate the number of Virtual Machines (VMs) required by a Software-as-a-Service (SaaS) provider using an Infrastructure-as-a-Service (IaaS) backend. Bodík *et al.* [2] use statistical machine learning to estimate the workload in the next epoch. Like other approaches, this requires additional servers to be provisioned in case the estimate is low.

Another static version of the server hiring problem was considered by Lampe *et al.* [9], who examined the optimal placement of a fixed set of jobs, with given run times and resource requirements, onto different Cloud servers. An exact formulation based on Binary Integer Programming and an approximate algorithm

[1] `http://www.cyclecomputing.com`

using bin-packing techniques were proposed. A similar problem involving workflows was addressed by Byun *et al.* [3,4]. In this instance, the servers are not different, but the jobs must satisfy a set of precedence constraints. Again, the aim is to minimize the cost of executing a given workflow on the Cloud. An approximate scheduling algorithm is proposed.

Chaisiri *et al.* [6] attempt to exploit the lower costs of future reservations in order to minimize the overall cost of hiring Cloud resources. They use stochastic and deterministic programming techniques, coupled with sample-average approximations or Benders decomposition. This study has some dynamic aspects. However, the actual demand process is not modelled and therefore the costs of waiting cannot be taken into account.

The server hiring problem is distantly related to other server allocation topics, for which a large body of literature exists. These topics include the trade-offs between performance and power consumption in a service center. In Mazzucco *et al.* [11] and Mitrani [13], certain dynamic server allocation policies were analysed, but no attempt was made to find the optimal policy. The maximization of throughput and the minimization of waiting or response time were considered in Urgaonkar *et al.* [17], Chandra *et al.* [5] and Bennani and Menascé [1].

The general Semi-Markov decision process and the algorithm for computing the optimal policy are described in section 2. The applications of the theory to the models with batch arrivals and with fixed hiring periods are presented in sections 3 and 4, respectively. Section 5 introduces the heuristic and shows the results of some numerical experiments. Some directions for further research are outlined in the conclusion – section 6.

2 Semi-Markov Decision Processes

Consider a finite-state system which is observed at random points in time, t_i ($i = 0, 1, \ldots$). These instants are called 'decision epochs' and the intervals between them are 'decision intervals'. If at time t_i the system is in state j ($j = 1, 2, \ldots, J$), an action, or decision, a_j, is taken. That action may influence the length of the ensuing decision interval, $t_{i+1} - t_i$, and also the system state at the next epoch. However, neither the decision interval nor the next state depend on anything that happened prior to t_i. Such a process is called a 'semi-Markov decision process'. The actions taken in the various states constitute a 'stationary policy', if for all states j, whenever the state j is observed, the same action, a_j, is taken, regardless of current time and past history.

The system incurs costs which depend on the states it passes through and on the decisions taken in those states. Let $Z_A(t)$ be the total cost incurred up to time t under a stationary policy A. Then the long-run average cost of policy A per unit time is defined as the limit:

$$g(A) = \lim_{t \to \infty} \frac{1}{t} E[Z_A(t)] \; . \tag{1}$$

That quantity, which does not depend on the initial state, is the optimization criterion. The object is to find a policy A that minimizes $g(A)$.

The evolution of the process under the control of a stationary policy A is governed by the succession of states at decision epochs, the decisions made at those epochs and the costs incurred during the decision intervals. Let $p_{j,k}(A)$ be the transition probability that the system will be in state k at the next decision epoch, given that the current state is j and the policy is A; $j, k = 1, 2, \ldots, J$. Also, denote by $c_j(A)$ the average cost incurred during a decision interval, given the current state j and policy A. Finally, let $\tau_j(A)$ be the average length of the decision interval, given the current state and policy.

The long-run average cost of policy A, $g(A)$, can be computed by introducing certain quantities called 'relative values', v_j, $j = 1, 2, \ldots, J$, (Tijms [16]). These relative values, together with $g(A)$, satisfy a set of simultaneous linear equations:

$$v_j = c_j(A) - \tau_j(A)g(A) + \sum_{k=1}^{J} p_{j,k}(A)v_k \;\; ; \;\; j = 1, 2, \ldots, J \; , \tag{2}$$

with $c_j(A)$, $p_{j,k}(A)$ and $\tau_j(A)$ as defined above.

In this set, there are J equations with $J + 1$ unknowns. However, if the same constant, c, is added to all relative values v_j, the value of $g(A)$ would not change (since for each j, the sum of $p_{j,k}(A)$ with respect to k is 1). Therefore, the solution of (2) can be made unique by choosing an arbitrary state, m, and setting $v_m = 0$. The optimal policy can be determined by the following 'policy improvement' algorithm.

1. Choose some stationary policy A.
2. Compute $g(A)$ and v_j by solving (2).
3. For each j, find action a^* that minimizes the right-hand side of equation (2):

$$\min_a \left[c_j(A) - \tau_j(a)g(A) + \sum_{k=1}^{J} p_{j,k}(a)v_k \right] \; ,$$

   where $g(A)$ and v_k keep the values already computed.
4. If new actions a^* are the same as the old ones for all states, i.e. new policy A^* is the same as A, stop. Otherwise repeat from step 2, replace A with A^*.

This algorithm terminates after a finite number of iterations, producing an optimal policy and the corresponding long-run average cost, g.

An efficient and stable method for solving the set of equations (2) is to use Gauss-Seidel iterations, starting with $v_j = 0$ for all j. Convergence is assured because the coefficients in the right-hand sides of (2), being probabilities, do not exceed 1. If that method is adopted, then the complexity of computing the optimal policy is on the order of $O(J^2SI)$, where J is the size of the state space, S is the number of iterations in the Gauss-Seidel solution and I is the number of iterations in the policy-improvement algorithm.

3 Batch Arrivals

The first system we examine is one where user demands arrive at the host's site in a Poisson stream with rate λ. Consecutive demands consist of batches of jobs

whose sizes are i.i.d. random variables with an arbitrary distribution. Let b_j be the probability that a batch contains j jobs ($j = 1, 2, \ldots, ...$). The average batch size is denoted by b:

$$b = \sum_{j=1}^{\infty} j b_j \ . \tag{3}$$

A job's runtime, on any available server, is distributed exponentially with mean $1/\mu$. Thus, the total offered load at the site is $\rho = \lambda b/\mu$. When all available servers are busy, jobs wait in a common FIFO queue. Servers may be hired and released at any moment.

In this model, the decision epochs are the instants just after the arrival of a new batch. The system state at a decision epoch is the total number, j, of jobs present. That number may include jobs from previous batches that are still waiting or are in service. The decision taken at a decision epoch is the number of servers, n, that are hired from a Cloud provider and will be available to serve jobs. That number may include previously hired servers, plus any newly hired ones, or minus any servers whose hire is terminated at this decision epoch.

Each job present incurs a holding cost of c_1 per unit time spent in the system. These costs reflect the importance attached to fast service. In addition, each hired server incurs a cost of c_2 per unit time. This is predicated on the assumption that the host is dealing with a Cloud that allows hire and release at arbitrary moments, with charges proportional to the duration of hire. A different hire regime will be modeled in the next section.

Thus, the total cost incurred per unit of time during which there are j jobs present and n servers hired is $c_1 j + c_2 n$.

Note that in this model the decision interval does not depend on the current state or on the decision taken. The average length of that interval is the average interarrival time between batches: $\tau = 1/\lambda$.

Since the algorithms available for determining the optimal policy require that the state space is finite, we assume that there is an upper bound, J, for the number of jobs that may be present. If an incoming batch would cause that bound to be exceeded, some or all of its jobs are rejected. That condition is not too restrictive: under any policy that does not allow the queue to saturate, one can choose J sufficiently large so that the probability of rejecting jobs is negligible. However, the numerical complexity of the solution increases with J.

To write equations (2) for a given policy A in the present model, we need expressions for $c_j(n)$ and $p_{j,k}(n)$, where n is the number of servers hired in state j under policy A. We start with the costs. Let $T_j(n)$ be the total average time that the j jobs currently present spend in the system during the decision period, given that n servers are available to serve them. There are two cases to consider:

1. If $j \leq n$, all jobs present are being served. The contribution of each job to $T_j(n)$ is the average minimum of its remaining service time and the remaining decision period. Hence, in this case,

$$T_j(n) = \frac{j}{\lambda + \mu} \ \ ; \ \ j = 1, 2, \ldots, n \ . \tag{4}$$

2. If $j > n$, then n jobs are being served and $j - n$ are waiting. The next event to occur is either a service completion, with probability $n\mu/(\lambda + n\mu)$, or an arrival of a new batch, with probability $\lambda/(\lambda+n\mu)$. The average interval until that event is $1/(\lambda + n\mu)$, and there are j jobs present during it. If the next event is a service completion, then the decision period continues with $j - 1$ jobs present; otherwise it terminates and there is no further contribution to $T_j(n)$. This provides a recurrence relation,

$$T_j(n) = \frac{j}{\lambda + n\mu} + \frac{n\mu}{\lambda + n\mu} T_{j-1}(n) \ ;$$

$$j = n + 1, n + 2, \ldots, J \ . \tag{5}$$

Equation (4), together with the recurrences (5), allow the holding times $T_j(n)$ to be computed easily for all j and n. The average cost, $c_j(n)$, incurred during a decision period is the sum of the holding cost and the server cost:

$$c_j(n) = c_1 T_j(n) + c_2 n \frac{1}{\lambda} \ . \tag{6}$$

Before addressing the transition probabilities $p_{j,k}(n)$, consider the probability, $q_{j,k}(n)$, that there will be k jobs present *just before* the next decision epoch, given that there are j jobs now and n servers are available. That is the probability that $j - k$ jobs are completed during the decision interval. There are three distinct cases:

1. If $j < n$, more servers become idle with each departing job. In order that k jobs are left at the end of the decision period, the latter must terminate when there are k busy servers. Hence,

$$q_{j,k}(n) = \left[\prod_{i=k+1}^{j} \frac{i\mu}{\lambda + i\mu} \right] \frac{\lambda}{\lambda + k\mu} \ ; \ \ k = 0, 1, \ldots, j \ , \tag{7}$$

   where an empty product is equal to 1 by definition.
2. If $j \geq n$ and $k \geq n$, then $q_{j,k}(n)$ is the probability that exactly $j - k$ jobs are completed by n busy servers before the decision period terminates:

$$q_{j,k}(n) = \left[\frac{n\mu}{\lambda + n\mu} \right]^{j-k} \frac{\lambda}{\lambda + n\mu} \ ; \ \ k = n, n + 1, \ldots, j \ . \tag{8}$$

3. If $j \geq n$ and $k < n$, then of the $j-k$ completions that must take place before the end of the observation period, $j - n + 1$ occur while n servers are busy and $n - 1 - k$ with gradually diminishing number of busy servers:

$$q_{j,k}(n) = \left[\frac{n\mu}{\lambda + n\mu} \right]^{j-n+1} \left[\prod_{i=k+1}^{n-1} \frac{i\mu}{\lambda + i\mu} \right] \frac{\lambda}{\lambda + k\mu} \ ;$$

$$k = 0, 1, \ldots, n - 1 \ . \tag{9}$$

Now we can obtain the transition probabilities from state j to state k, $p_{j,k}(n)$, by remarking that the number of jobs present after the arrival of the next batch is the convolution of the number left over at the end of the decision interval and the number contained in the new batch. Hence,

$$p_{j,k}(n) = \sum_{i=0}^{m} q_{j,i}(n) b_{k-i} \;\; ; \;\; k = 1, 2, \ldots, J-1 \; , \tag{10}$$

where $m = \min(j, k-1)$. The exception to that pattern is destination state J, which may be reached after rejecting some new arrivals:

$$p_{j,J}(n) = \sum_{i=0}^{j} q_{j,i}(n) \sum_{s=J-i}^{\infty} b_s \; . \tag{11}$$

All quantities necessary for setting up equations (2), and hence for applying the policy improvement algorithm, are now available.

N.B. The reason for assuming that the batch interarrival intervals are distributed exponentially was the tractability of the expressions for $c_j(n)$ and $p_{j,k}(n)$. It would be possible to relax that assumption, e.g. by replacing the exponential with a phase-type distribution. However, the resulting expressions would be considerably more complicated.

4 Fixed Hiring Periods

We now address a system where a server must be hired for a sizeable minimum period of time, τ. Amazon, for example, hires servers by the hour. Although in principle one could initiate a hire at any time, it is reasonable, and more tractable, to use the instants 0, τ, 2τ, ..., as decision epochs (i.e., the length of the decision interval is τ). Assume that jobs arrive singly during a decision interval, in a Poisson stream with rate λ. Their service times are again distributed exponentially, with mean $1/\mu$.

Thus, if there are j jobs in the system at a decision epoch, and n servers are hired, then during an interval of length τ the queue behaves as a transient $M/M/n/J$ queue (J is the bound on the number of jobs present), with initial state j. To define our decision process, we need the transition probabilities, $p_{j,k}(n)$, that there will be k jobs at time τ, given that there were j jobs at time 0 and n servers were hired.

Denote by $P(t) = [p_{j,k}(t)]$, $j, k = 0, 1, \ldots, J$, the transient transition probability matrix for the $M/M/n/J$ queue over the interval (0,t). Clearly, $P(0) = I$, where I is the $(J+1) \times (J+1)$ identity matrix. We are interested in computing the j'th row of $P(\tau)$.

Let G be the generator matrix for the $M/M/n/J$ queue:

$$G = \begin{bmatrix} -\lambda & \lambda & & & & \\ \mu_1 & -(\lambda+\mu_1) & \lambda & & & \\ & & \ddots & & & \\ & & & -(\lambda+\mu_{J-1}) & \lambda & \\ & & & \mu_J & -\mu_J \end{bmatrix} , \tag{12}$$

where $\mu_i = \min(i, n)\mu$. The matrix $P(t)$ is given by the matrix-exponential:

$$P(t) = e^{Gt} . \tag{13}$$

If the solution algorithms are implemented in Matlab, this matrix exponentiation can be performed by the built-in function $expm(G * t)$, which is stable and fast. If that is not available, one could employ the 'uniformization' technique, which involves replacing the continuous-time Markov process with an equivalent discrete-time Markov chain using the parameter $\gamma = \lambda + n\mu$ (e.g., see [14]). The generator matrix G is replaced by the matrix:

$$Q = I + \frac{G}{\gamma} ,$$

where I is the identity matrix. Then $P(t)$ is given by the series:

$$P(t) = \sum_{i=0}^{\infty} Q^i \frac{(\gamma t)^i}{i!} e^{-\gamma t} . \tag{14}$$

This expression provides an efficient way of computing $P(t)$ because (a) Q is a stochastic matrix, so the elements of Q^i remain uniformly bounded for all i (since the rows always sum up to 1), and (b) the Poisson probabilities that appear in (14) converge rapidly to 0. Hence, the infinite series can be truncated on the right, and possibly on the left, resulting in a finite sum:

$$P(t) = \sum_{i=\ell}^{r} Q^i \frac{(\gamma t)^i}{i!} e^{-\gamma t} , \tag{15}$$

where ℓ and r are chosen so that the two omitted tails are negligible (see [7]).

It remains to determine the average cost, $c_j(n)$, incurred during a decision interval. Let L_j be the average number of jobs in the system at time τ, given that there were j jobs at time 0 and n servers were hired. That average is obtained:

$$L_j = \sum_{k=1}^{J} k p_{j,k}(\tau) . \tag{16}$$

The average number of jobs present *during* the decision interval can be approximated by taking the mean of the queue sizes at the beginning and end of the interval, i.e. $(j + L_j)/2$. Hence, the total cost incurred during the interval is given by:

$$c_j(n) = \left[c_1 \frac{j + L_j}{2} + c_2 n \right] \tau \ . \tag{17}$$

Using these expressions, the optimal policy can be computed as described in section 2.

N.B. One might wish to relax the assumptions that jobs arrive in a Poisson stream during a decision interval, and their lengths are distributed exponentially. Some generalizations using phase-type distributions could be treated numerically, but replacing the $M/M/n/J$ queue with a $GI/G/n/J$ one would require major approximations.

5 Heuristics and Experiments

When the computation of the optimal becomes expensive, it may be worth exploring policies that are sub-optimal, but offering good performance and ease of implementation.

A promising heuristic policy for any given model is the one which, at every decision epoch, minimizes the average cost incurred during the current decision interval. In other words, when the current state is j, take the action n^* such that:

$$c_j(n^*) = \min_n c_j(n) \ , \tag{18}$$

where $c_j(n)$ is the cost appropriate to the model. This short-term policy that looks only at the current state and does not care about the future. It will be called the 'greedy' heuristic, as this type of policies are commonly referred to.

The implementation of the greedy heuristic does not require any iterations; it is enough to evaluate the costs $c_j(n)$ for different values of n. Hence, the complexity of implementing the greedy heuristic is $O(JC)$, where C is the complexity of evaluating an individual cost. In practice, the greedy heuristic is orders of magnitude faster to find than the optimal policy.

The performance of the greedy heuristic will be compared with that of the optimal policy, for each of our models. In addition, an even simpler policy will be introduced to use as a benchmark. The latter abandons dynamic decision-making altogether and hires a fixed number of servers, n^*, regardless of the system state. This is, in fact, the policy often adopted in practice. To avoid saturating the queue, n^* should be chosen so that the average long-term server occupancy is less than 100%. For example, one could aim for an occupancy of 70%. In the case of batch arrivals, bearing in mind that the offered load is $\rho = \lambda b/\mu$, where b is the average batch size, the above condition implies:

$$n^* = \left\lceil \frac{\lambda b}{0.7\mu} \right\rceil \ . \tag{19}$$

For the second model, the offered load is $\rho = \lambda/\mu$, so the allocation becomes:

$$n^* = \left\lceil \frac{\lambda}{0.7\mu} \right\rceil \ . \tag{20}$$

That policy will be referred to as the 'fixed policy'.

Figure 1 illustrates and compares the behaviour of the three policies for the batch arrivals model, in the case where batch sizes are distributed geometrically with parameter α. That is, the probability that a batch contains j jobs is $\alpha(1-\alpha)^{j-1}$. The average batch size is $b = 1/\alpha$. The offered load is increased by decreasing α, and the long-term average cost, g, is plotted against the average batch size. The average service time is $1/\mu = 1$, while the batch arrival rate is $\lambda = 0.1$. In this experiment, it was assumed that the unit holding cost and the unit server cost are equal: $c_1 = c_2 = 1$.

The bound on the number of jobs in the system was taken as $J = 100$. Under all three policies, the probability of reaching that bound is small. For example, when the average batch size is 50, the probability that a batch of size 100 will be submitted is about 0.1.

A notable feature of the figure is that the greedy heuristic is almost optimal over the entire range of offered loads. One would therefore be justified in using the heuristic in practice, knowing that its performance cannot be improved significantly. By contrast, the costs of the fixed policy are considerably higher. That remains the case if the 70% occupancy of the servers is replaced by 80% occupancy. Of course, the more the fixed policy over-provides servers unnecessarily, the poorer its performance would be. The non-monotone character of the graph for the fixed policy is due to the rounding-up operation in (19).

Next, we experiment with a batch size distribution that has been constructed to have a large coefficient of variation. More precisely, batches consist of a single job with probability 0.7, and B jobs with probability 0.3. The average batch size is $b = 0.7 + 0.3B$. The coefficient of variation grows roughly linearly with B. In figure 2, B is varied between 20 and 100, and the average achieved cost is plotted against b.

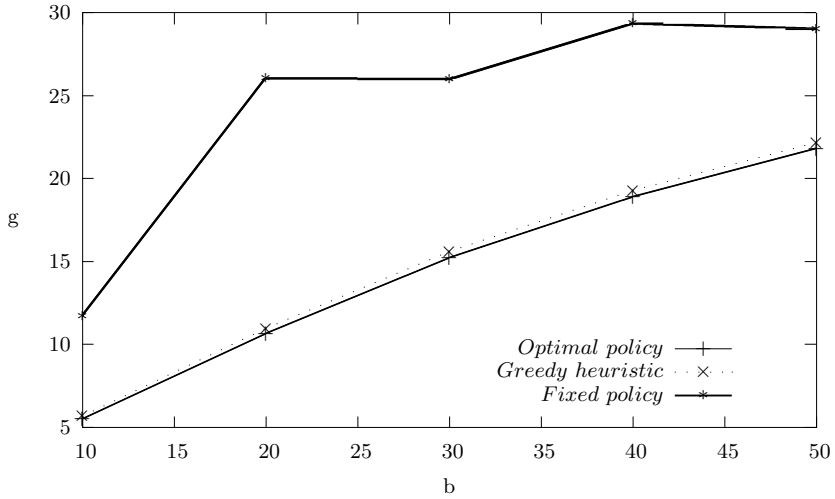

Fig. 1. Batch arivals: geometric batch sizes

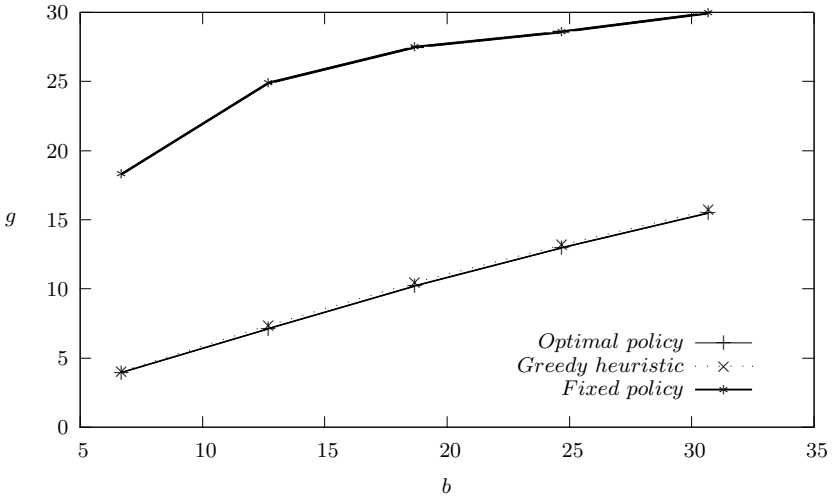

Fig. 2. Batch arrivals: skewed batch size distribution

It seems that large coefficients of variation do not prevent the greedy heuristic from performing well. Its costs are almost indistinguishable from those of the optimal policy. On the other hand, the fixed policy is, if anything, worse than before in comparison.

In the third experiment, the characteristics of the demand are held fixed, at $\lambda = 0.1$, $\mu = 1$, $\alpha = 0.04$ (i.e., average batch size of 25). Also, the unit server cost is fixed at $c_2 = 10$. What is varied is the unit holding cost, from $c_1 = 5$ to $c_1 = 20$. That is, the relative cost of keeping jobs in the system is varied from half to double the cost of a server.

The results are shown in figure 3, where the average long-term costs g achieved by the optimal policy, the greedy heuristic and the fixed policy are plotted against c_1.

Again, it is notable that the greedy heuristic achieves nearly optimal costs over the entire range of c_1 values. By contrast, the performance of the fixed policy is rather poor. Moreover, whereas the cost of the fixed policy grows linearly with c_1 (as can be expected), those of the optimal and greedy policies grow slower than linearly.

It is perhaps worth pointing out that, for all points in these three figures, the policy improvement algorithm took no more than 3 iterations to find the optimal policy.

The remaining experiments concern the model with fixed hire periods. In figure 4, the offered load is increased from $\rho = 10$ to $\rho = 18$ by varying the job arrival rate. The service rate is kept at $\mu = 1$, and the unit holding cost is half of the server cost: $c_1 = 0.5$, $c_2 = 1$. The bound on the number of jobs is $J = 50$. The hire period length is $\tau = 4$, meaning on average, between 40 and 72 jobs arrive during a decision period. The fixed policy is based on equation (20).

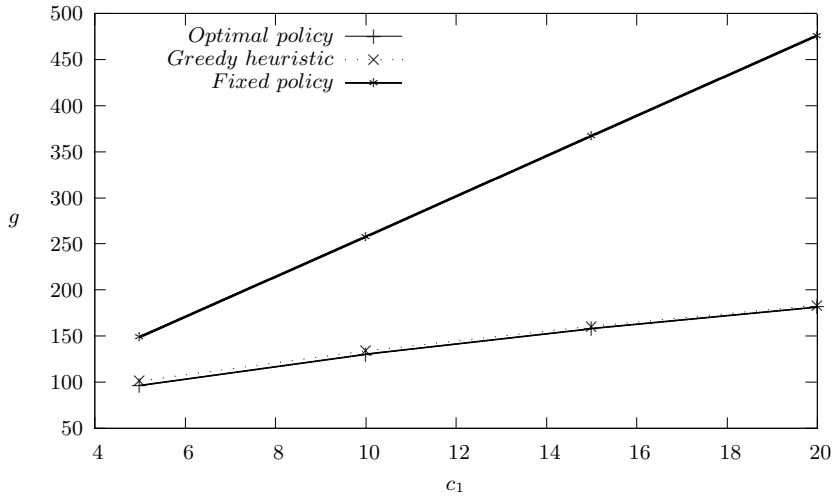

Fig. 3. Batch arrivals: varying unit holding cost

We observe that the difference between the worst policy (fixed) and the best one (optimal) is now much narrower. This is due to the fact that jobs continue to arrive throughout a decision period, and the rate of arrivals does not depend on the action taken. This reduces the advantages derived from making dynamic decisions. The costs achieved by the optimal policy are about 15% lower than those of the fixed policy. The greedy heuristic still performs quite well, but its costs are now about 10% higher than those of the optimal policy.

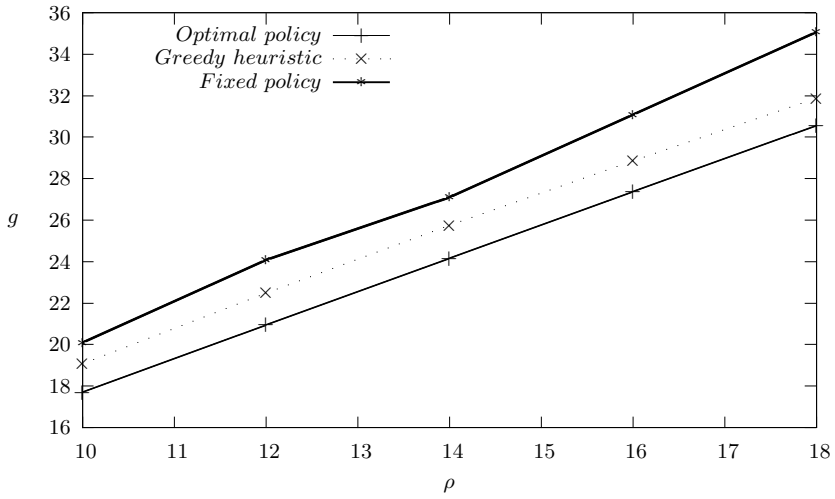

Fig. 4. Fixed hiring periods: varying offered load

In figure 5, the job arrival rate is kept fixed at $\lambda = 12$. The service rate, server cost and decision period length have the same values as before, $\mu = 1$, $c_2 = 1$, $\tau = 4$, while the unit holding cost is varied from half to twice the server cost: $0.5 \leq c_1 \leq 2$.

The average costs achieved by the three policies are quite close over the entire range of c_1 values. Moreover, it is notable that the higher the value of c_1 relative to c_2, the closer those costs are, i.e. the lower the benefit of dynamic decision-making. Indeed, one could have expected that when the dominant factor

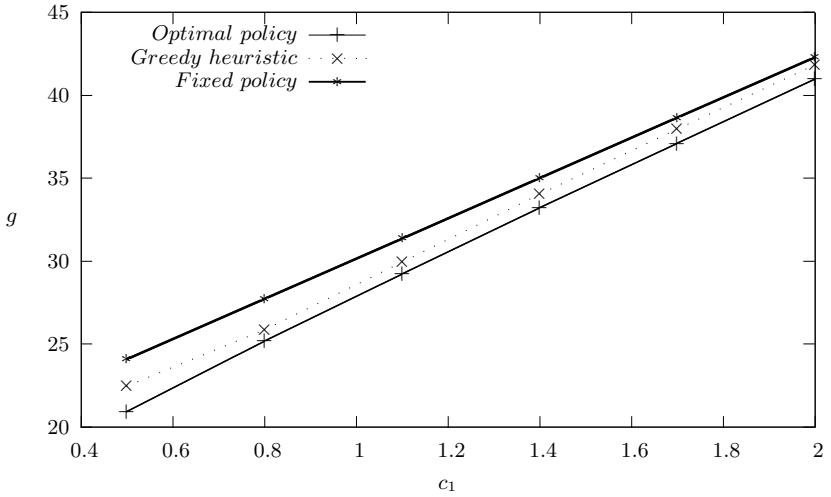

Fig. 5. Fixed hiring periods: varying unit holding cost

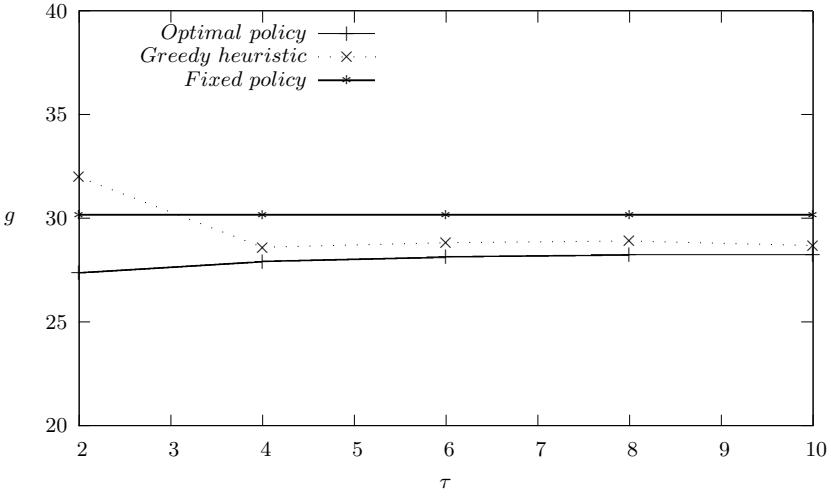

Fig. 6. Fixed hiring periods: varying hire period length

is customer performance, the most important part of the policy is to always maintain enough servers to cope with the load.

In the final experiment, traffic characteristics and unit costs are kept fixed ($\lambda = 12$, $\mu = 1$, $c_1 = c_2 = 1$), while the length of the decision interval is varied from $\tau = 2$ to $\tau = 10$. That is, the average number of arrivals during a decision interval varies from 24 to 120.

The fixed policy is independent of τ, so its graph is a horizontal line. The optimal and greedy policies also approach a horizontal asymptote. This is predictable, since the system tends to reach steady state during a large decision interval, and the distribution at the next decision epoch becomes independent of the current state. For the same reason, the greedy heuristic, whose performance can be worse than that of the fixed policy for very short decision intervals, becomes not only 'nearly optimal', but optimal, in the limit $\tau \to \infty$.

For all points in the last three figures, the policy improvement algorithm again took no more that 3 iterations to find the optimal policy.

6 Conclusions

The problem of minimizing costs in a system where servers are hired dynamically was considered in the context of two traffic and hiring regimes: batch arrivals with arbitrary hiring intervals and Poisson arrivals with fixed hiring intervals. In both cases, the optimal hiring policy can be computed by applying a policy improvement algorithm. In addition, greedy heuristic policies are available which are often almost indistinguishable from the optimal policy.

One can envisage extending the models in several directions. For example, there may be jobs of different types, with different arrival and service characteristics and different holding and server costs. The system state at a decision epoch would then be a vector $(j_1, j_2, \ldots, j_k)$, where j_i is the number of jobs of type i present. The action taken at a decision epoch would also be a vector of server allocations, $(n_1, n_2, \ldots, n_k)$, where n_i is the number of servers hired to serve jobs of type i. The methodology described here would still apply, but the computation of the optimal policy would be considerably more complex. Another generalization would be to allow the traffic parameters λ and μ to change between decision intervals. They may depend on the current state, and possibly on the action taken, or may be controlled by a changing environment. Such systems could also be handled by the methods proposed here.

References

1. Bennani, M.N., Menascé, D.: Resource allocation for autonomic data centers using analytic performance methods. In: Procs. 2nd IEEE Conf. on Autonomic Computing, ICAC 2005), pp. 229–240 (2005)
2. Bodík, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., Patterson, D.: Statistical machine learning makes automatic control practical for internet datacenters. In: Conf. on Hot Topics in Cloud Computing, HotCloud 2009, Berkeley, CA, USA (2009)

3. Byun, E.-K., Kee, Y.-S., Kim, J.-S., Maeng, S.: Cost optimized provisioning of elastic resources for application workflows. Future Generation Computer Systems 27(8), 1011–1026 (2011), http://dx.doi.org/10.1016/j.future.2011.05.001
4. Byun, E.-K., Kee, Y.-S., Kim, J.-S., Deelman, E., Maeng, S.: BTS: Resource capacity estimate for time-targeted science workflows. Journal of Parallel and Distributed Computing 71(6), 848–862 (2011), doi:10.1016/j.jpdc.2011.01.008
5. Chandra, A., Gong, W., Shenoy, P.: Dynamic resourse allocation for shared data centers using online measurements. In: Procs. 11th ACM/IEEE Int. Workshop on Quality of Service (IWQoS), pp. 381–400 (2003)
6. Chaisiri, S., Lee, B.S., Niyato, D.: Optimization of resource provisioning cost in cloud computing. IEEE Transactions on Services Computing 5(2), 164–177 (2012)
7. Fox, B.L., Glynn, P.W.: Computing Poisson Probabilities. Management Science and Operations Research 31(4), 440–445 (1988)
8. Hiden, H., Woodman, S., Watson, P., Cala, J.: Developing cloud applications using the e-science central platform. Royal Soc. of London, Phil. Trans. A. (Mathematical, Physical and Engineering Science), 371 (2013)
9. Lampe, U., Siebenhaar, M., Hans, R., Schuller, D., Steinmetz, R.: Let the clouds compute: Cost-efficient workload distribution in infrastructure clouds. In: Vanmechelen, K., Altmann, J., Rana, O.F. (eds.) GECON 2012. LNCS, vol. 7714, pp. 91–101. Springer, Heidelberg (2012)
10. Mazzucco, M., Dyachuk, D., Dikaiakos, M.: Profit-aware server allocation for green internet services. In: IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 277–284 (2010)
11. Mazzucco, M., Mitrani, I., Fisher, M., McKee, P.: Allocation and Admission Policies for Service Streams. In: Procs. MASCOTS 2008, Baltimore, pp. 155–162 (2008)
12. Mazzucco, M., Vasar, M., Dumas, M.: Squeezing out the cloud via profit-maximizing resource allocation policies. In: IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 19–28 (2012)
13. Mitrani, I.: Managing Performance and Power Consumption in a Server Farm. Annals of Operations Research (2011), doi:10.1007/s10479-011-0932-1
14. Reibman, A., Trivedi, K.: Numerical transient analysis of Markov models. Computing and Operations Research 15(1), 19–36 (1988)
15. D. Thain, T. Tannenbaum and Miron Livny, "Distributed computing in practice: the Condor experience", Concurrency and Computation: Practice and Experience, 17 (2-4),323-356, doi:http://dx.doi.org/10.1002/cpe.v17:2/4
16. Tijms, H.C.: Stochastic Models. John Wiley and sons (1994)
17. Urgaonkar, R., Kozat, U.C., Igarashi, K., Neely, M.J.: Dynamic Resource Allocation and Power Management in Virtualized Data Centers. In: IEEE/IFIP NOMS 2010, Osaka, Japan (2010)